Microsoft

Taller de introducción a IoT

Microsoft Developer eXperience

Contenido

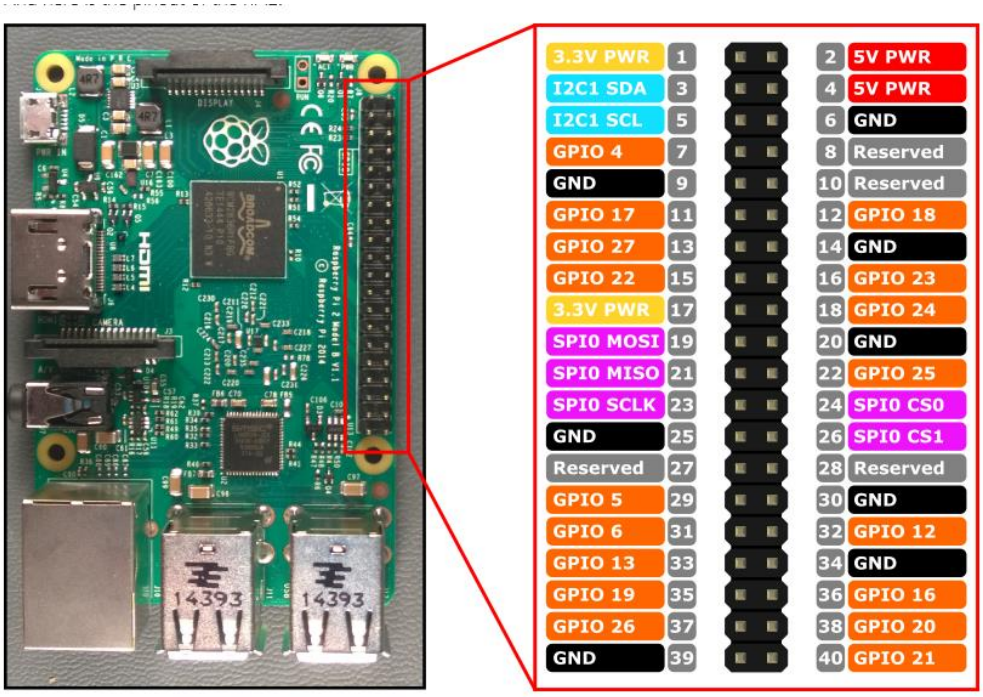
Introducción

Con Windows 10 IoT Core podemos desarrollar aplicaciones universales que se ejecutan en dispositivos de bajo consumo como la Raspberry Pi 2. En este taller vamos a introducirnos en el desarrollo de aplicaciones universales utilizando las características de los dispositivos con IoT Core como el bus GPIO.

Fase 1: Hacer parpadear un led

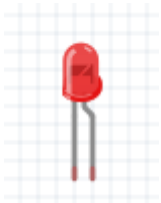
La primera parte del taller consistirá en desarrollar un programa que haga parpadear un led cada 500 milisegundos.

Trabajaremos con una Raspberry Pi 2, que es como un ordenador, pero en un espacio mucho más reducido. Esta placa tiene una serie de pines a los que conectaremos los elementos con los que vamos a trabajar. Cada pin se identifica con un número. Este número es lo que nos permite desde nuestro programa controlar los leds, botones… que conectaremos a nuestra Raspberry.

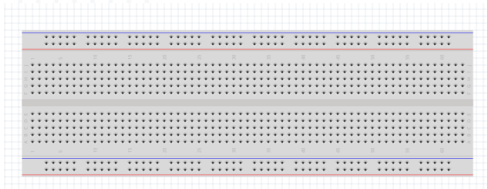


Para esta primera fase, aparte de la Raspberry Pi, vamos a necesitar:

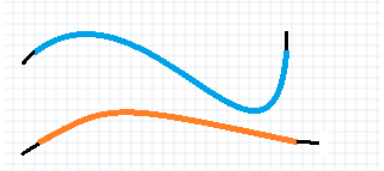
1 Led



1 Protoboard

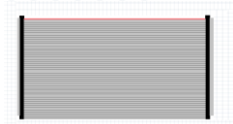


2 Cables

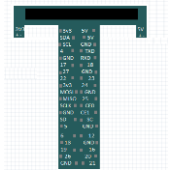


1 Resistencia

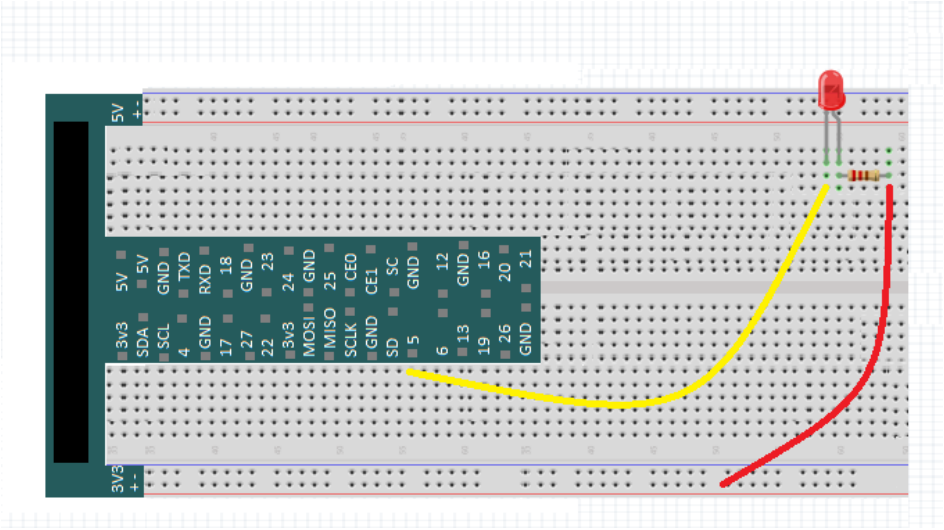
1 Bus GPIO



1 Breakout board



Montamos el circuito de la siguiente manera:



La pata corta del led la conectamos con el cable azul al pin GPIO 5.

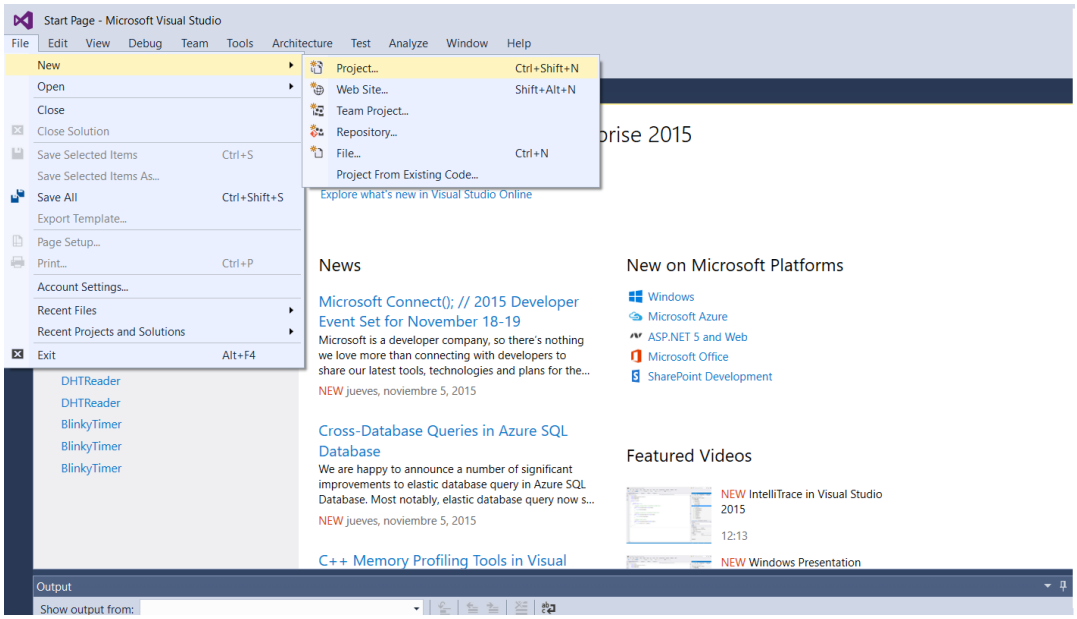
La pata larga la conectamos a la resistencia. Esta pata es la que recibe la corriente desde la Raspberry por lo que hemos de conectarla a la resistencia.

La resistencia la conectamos con el cable rojo al pin de corriente de 3.3 voltios. Es muy importante conectar bien la resistencia ya que es la que impedirá que el Led se funda.

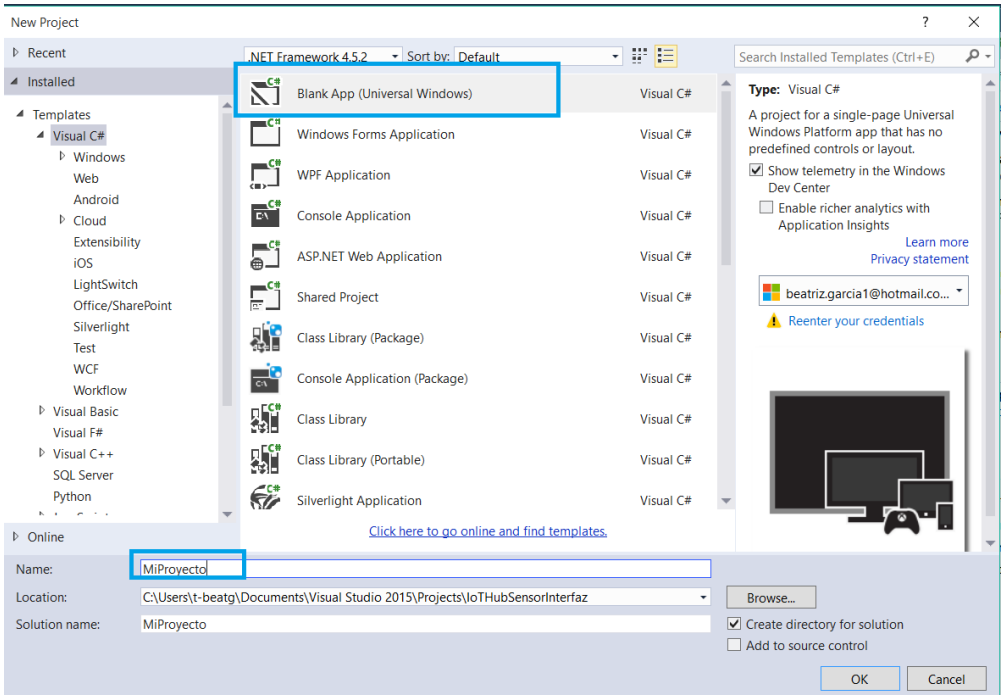
Una vez tenemos el circuito montado, hemos de conectar la Raspberry a la corriente y a la red. Al conectarla a la red se le asignará una IP. Esta IP equivale al DNI de nuestra Raspberry y nos permite identificarla para poder instalar en ella el programa que vamos a desarrollar.

Ahora vamos a abrir Visual Studio para escribir la aplicación que cargaremos en la Raspberry y que se encargará de hacer parpadear un led.

Accedemos a Visual Studio y pulsamos en File, New y luego en Project.

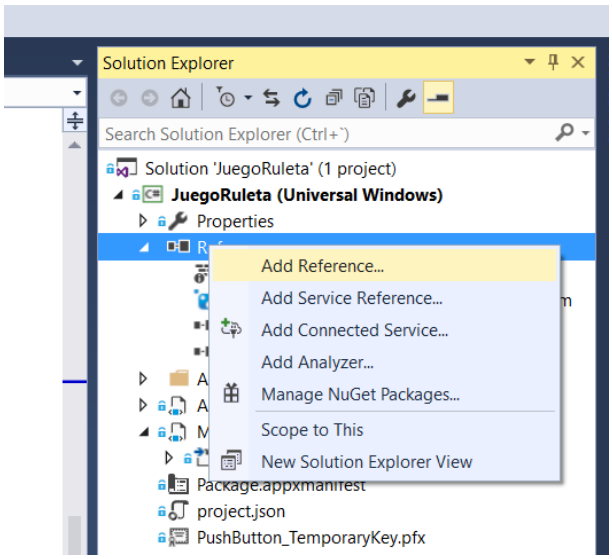


Se abre la ventana para seleccionar el tipo de proyecto y seleccionamos Blank App (Universal Windows). Asignamos al proyecto un nombre y pulsamos en OK.

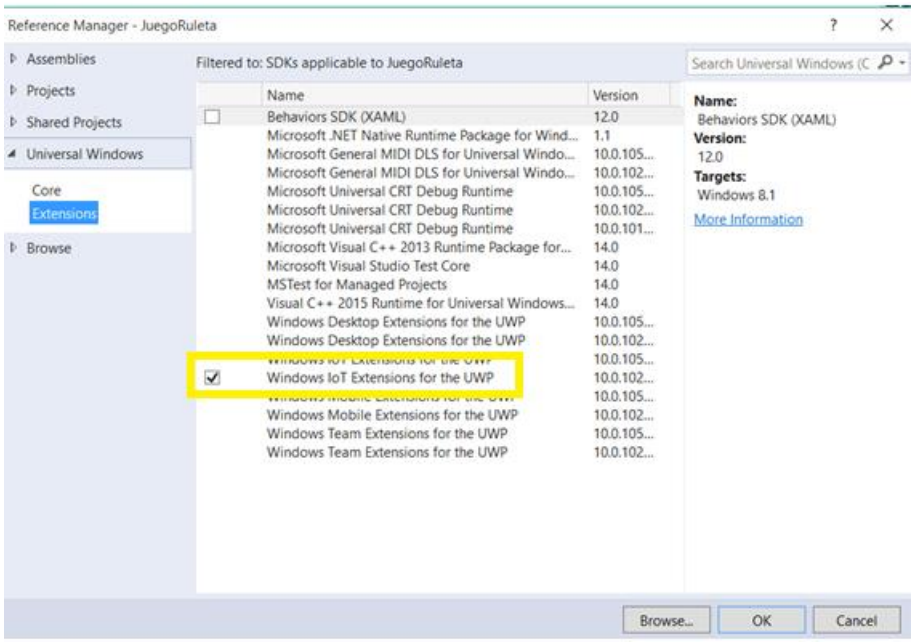


Tras unos segundos el proyecto se ha creado con el nombre elegido.

Para poder acceder a los pines de la Raspberry necesitamos poner las extensiones de IoT en nuestro proyecto. Para ello, pulsamos el botón derecho sobre la carpeta que pone “References” dentro del proyecto y seleccionamos “Add Reference”.

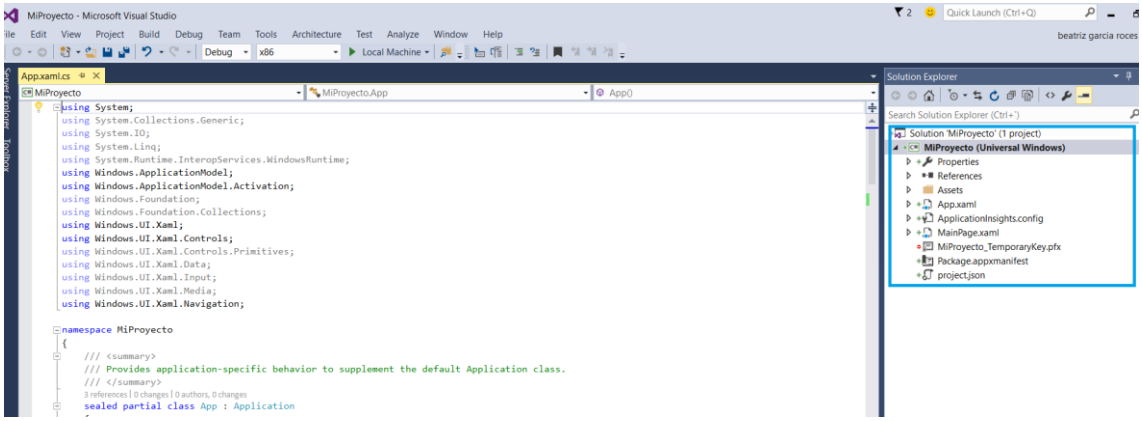


Cogemos las extensiones de IoT.

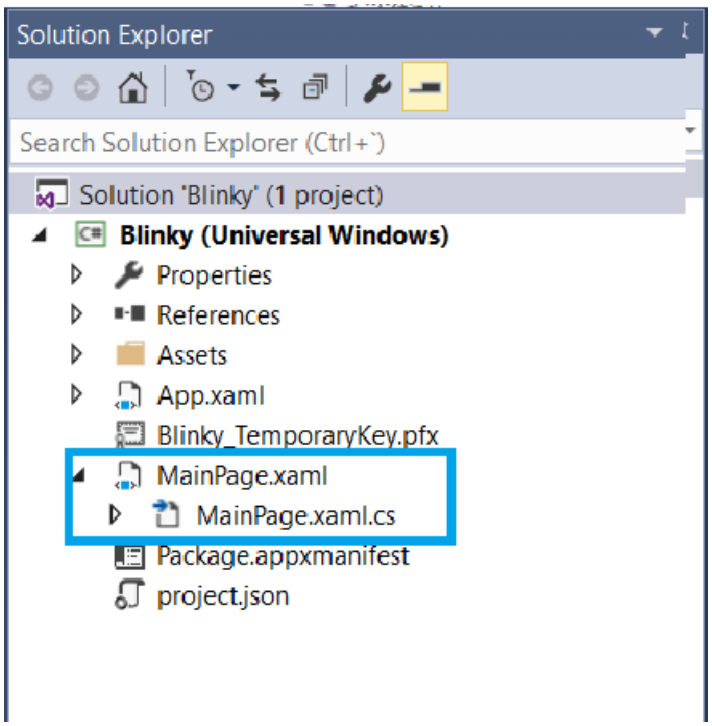


Y pulsamos en Ok.

La referencia se habrá añadido al proyecto.



Vemos que dentro del proyecto tenemos un archivo que se llama MainPage.xaml.cs, en esa clase es donde tenemos que introducir nuestro código.



Dentro de la clase y antes del método MainPage introducimos el siguiente código:

```
        private const int LED_PIN = 5;
        private GpioPin pin;
        private GpioPinValue pinValue;
        private DispatcherTimer timer;
```

Al introducir este fragmento del código veremos que GpioPin y GpioPinValue quedan subrayados en rojo, esto se debe a que necesitamos añadir un using para poder trabajar con los buses GPIO de la Raspberry:

```
         using Windows.Devices.Gpio;
```

Reemplazamos el método MainPage por el siguiente código encargado de inicializar el temporizador y los pines.

```
public MainPage()
        {
            InitializeComponent();
            timer = new DispatcherTimer();
            timer.Interval = TimeSpan.FromMilliseconds(500);
            timer.Tick += Timer_Tick;
            InitGPIO();
            if (pin != null)
            {
                timer.Start();
            }
        }
```

Las llamadas a los métodos Timer_Tick y InitGPIO se quedarán subrayadas en rojo ya que aun no hemos creado los métodos.

Creamos un método InitGPIO necesario para indicar en qué pin hemos conectado el led, si es un pin de entrada o de salida (en este caso al ser un led es de salida) y si queremos que comience encendido (low) o apagado (high).

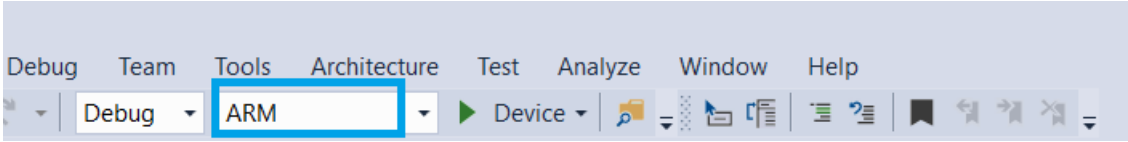
```
private void InitGPIO()
        {
            var gpio = GpioController.GetDefault();

            if (gpio == null)
            {
                pin = null;

                return;
            }

            pin = gpio.OpenPin(LED_PIN);
            pinValue = GpioPinValue.High;
            pin.Write(pinValue);
            pin.SetDriveMode(GpioPinDriveMode.Output);


        }
```

Creamos otro método llamado Timer_Tick que cada vez que pase el tiempo establecido en el temporizador (500 milisegundos) comprobará si el led está encendido o apagado. Si está encendido lo apagará y si está apagado lo encenderá.

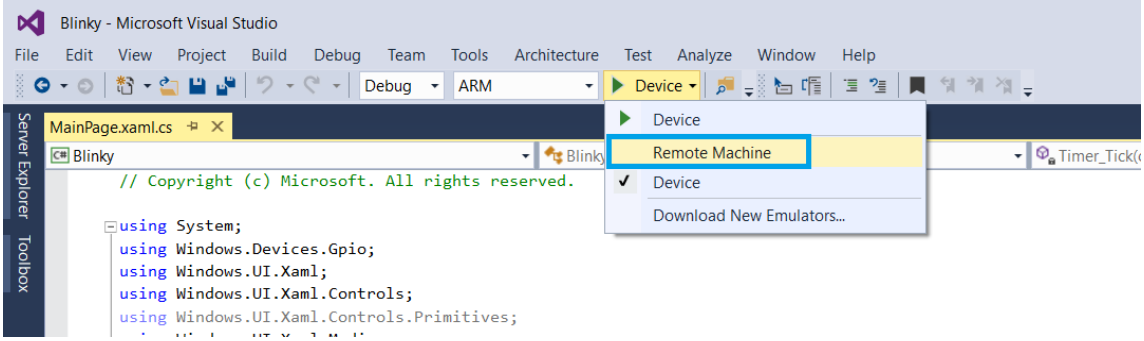
```
private void Timer_Tick(object sender, object e)
        {
            if (pinValue == GpioPinValue.High)
            {
                pinValue = GpioPinValue.Low;
                pin.Write(pinValue);

            }
            else
            {
                pinValue = GpioPinValue.High;
                pin.Write(pinValue);
            }
        }
```

Ahora vamos a ejecutar la aplicación. Para ello debemos desplegarla en la Raspberry.

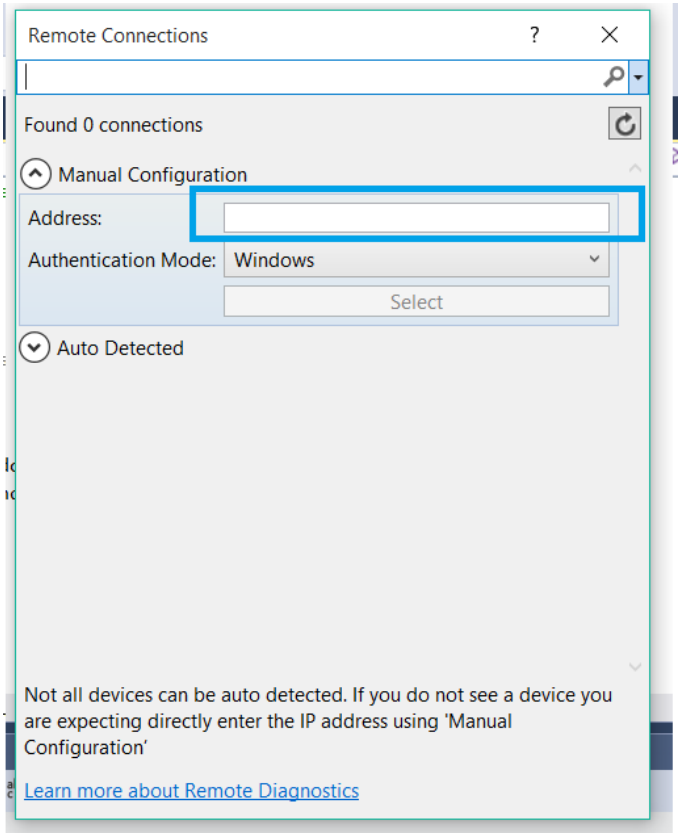
Lo primero que tenemos que hacer es cambiar la arquitectura para la que se va a compilar la aplicación. El procesador de la Raspberry Pi es ARM por lo que seleccionamos ARM en el menú superior.



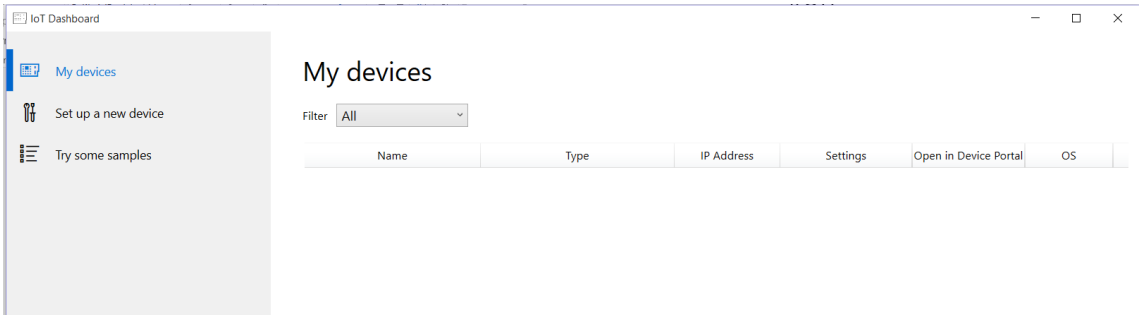
A continuación, seleccionamos Remote Machine, ya que queremos desplegar la aplicación en la Raspberry de forma remota.



Se abrirá una pantalla para introducir la IP de nuestra Raspberry.

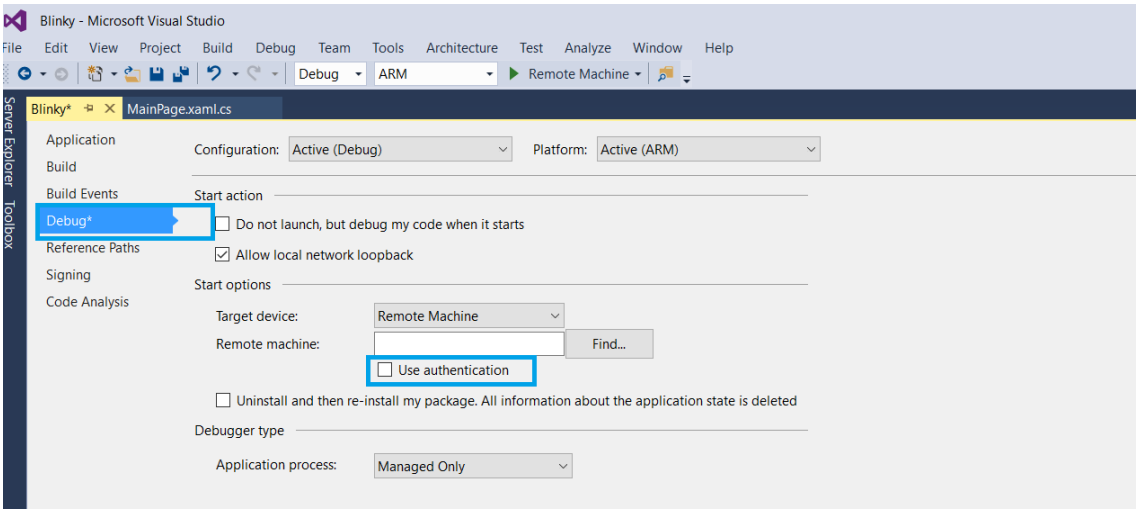


Para saber cuál es la IP de la Raspberry abrimos Windows 10 IoT Core Dashboard y en el apartado My Devices buscamos la IP de nuestra Raspberry.



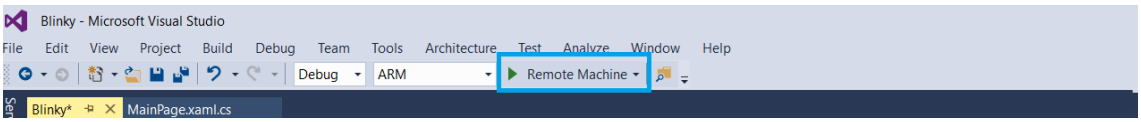
Volvemos a Visual Studio, ponemos la IP y cerramos la ventana. Para poder ejecutar nuestra aplicación hemos de hacer una cosa más.

Sobre el proyecto pulsamos el botón derecho de nuestro ratón y seleccionamos Properties. En el apartado Debug desmarcamos la casilla de Use authentication.



Pulsamos el botón CRTL y la S a la vez y se guardaran estos ajustes.

Ahora ya podemos darle a Remote Machine y veremos que tras unos segundos el led empieza a parpadear.

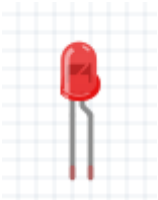


Fase 2: Encender un led con un botón

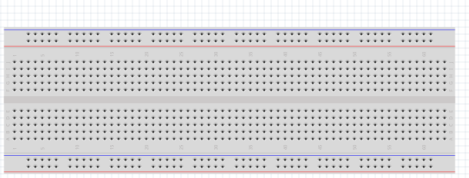
En esta fase del taller vamos a poner un botón en la protoboard que utilizaremos para encender y apagar el led.

Aparte de la Raspberry Pi, necesitaremos:

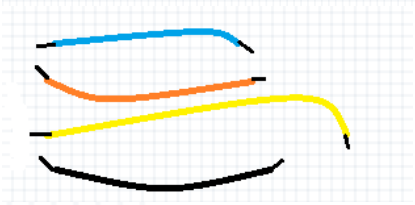
1 Led



1 Protoboard



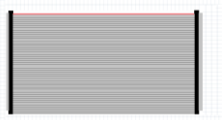
4 Cables



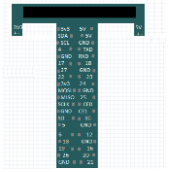
1 Resistencia

1 Botón

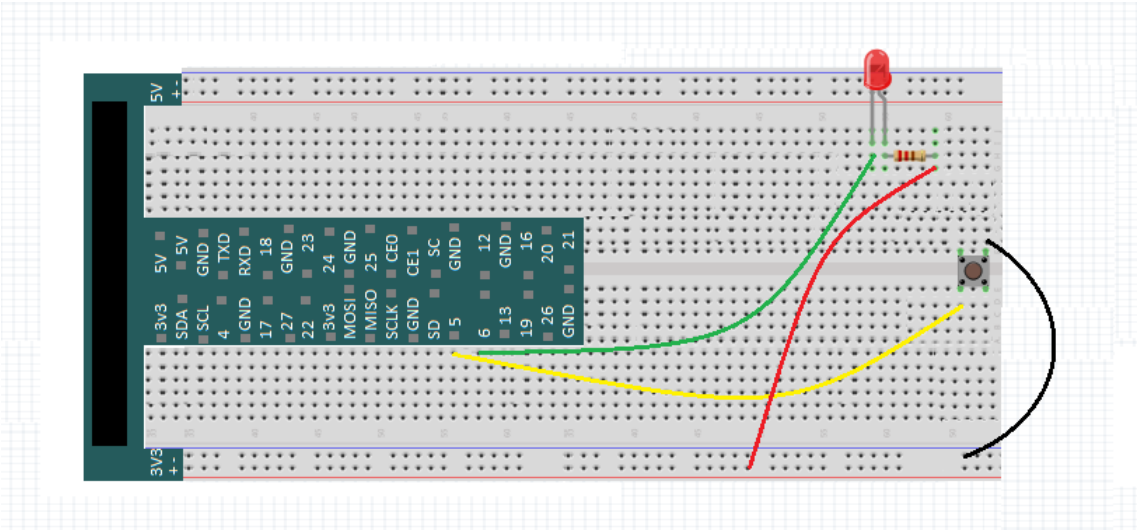
1 Bus GPIO



1 Breakout board



Montamos el circuito de la siguiente manera:



La pata larga del led a la resistencia.

La pata corta del led al GPIO 6.

La resistencia a la corriente, al carril rojo 3V.

Una de las patas del botón a la toma de tierra, el carril azul.

Otra de las patas del botón al GPIO 5.

Ahora conectamos la Raspberry a la corriente de nuevo.

Abrimos Visual Studio para programar el código que va a hacer que al pulsar el botón se encienda y apague el led.

Creamos un nuevo proyecto del mismo tipo que el anterior pero con otro nombre.

Al igual que en el caso anterior, dentro de la clase MainPage.xaml.cs antes del método MainPage introducimos

```
        private const int LED_PIN = 6;
        private const int BUTTON_PIN = 5;
        private GpioPin ledPin;
        private GpioPin buttonPin;
        private GpioPinValue ledPinValue = GpioPinValue.High;
```

Hemos de añadir de nuevo la directiva using.

```
        using Windows.Devices.Gpio;
```

Sustituimos el método MainPage por el siguiente, encargado de llamar al método que inicializa los pines.

```
        public MainPage()
        {
            InitializeComponent();
            InitGPIO();

        }
```

A continuación, creamos el método InitGPIO que se encarga de abrir los pines, establecer si son de salida (led) o de entrada (botón) y darles el valor inicial.

```
private void InitGPIO()
        {
            var gpio = GpioController.GetDefault();
            if (gpio == null)
            {
                return;
            }

            buttonPin = gpio.OpenPin(BUTTON_PIN);
            ledPin = gpio.OpenPin(LED_PIN);
```

```
            ledPin.Write(GpioPinValue.High);
            ledPin.SetDriveMode(GpioPinDriveMode.Output);

            if (buttonPin.IsDriveModeSupported(GpioPinDriveMode.InputPullUp))
                buttonPin.SetDriveMode(GpioPinDriveMode.InputPullUp);
            else
                buttonPin.SetDriveMode(GpioPinDriveMode.Input);


            buttonPin.DebounceTimeout = TimeSpan.FromMilliseconds(50);


            buttonPin.ValueChanged += buttonPin_ValueChanged;


        }
```

Creamos también el método buttonPin_ValueChanged que se encarga de, cuando se pulsa el botón, comprobar el estado del led, si está encendido lo apaga y si está apagado lo enciende.

```
private void buttonPin_ValueChanged(GpioPin sender, GpioPinValueChangedEventArgs
e)
        {

            if (e.Edge == GpioPinEdge.FallingEdge)
            {
                ledPinValue = (ledPinValue == GpioPinValue.Low) ?
                    GpioPinValue.High : GpioPinValue.Low;
                ledPin.Write(ledPinValue);
            }


        }
```

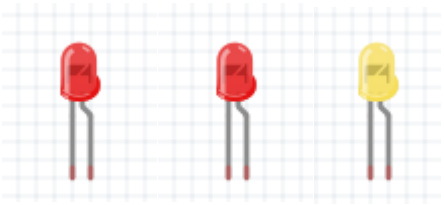
Desplegamos el código en la Raspbery siguiendo los mismos pasos que en la fase 1.

Fase 3: Juego de la ruleta

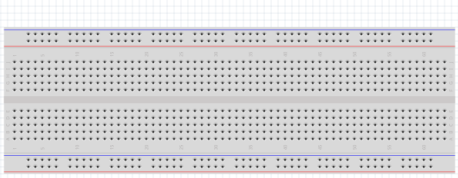
En esta última fase del taller haremos un juego con 3 leds. Tendremos dos rojos y un amarillo. Se irán encendiendo en orden, primero el amarillo y luego los rojos y así constantemente. El juego consiste en que hay que darle al botón cuando el led que esté encendido sea el amarillo, de forma que el movimiento de luces se pare y el led amarillo se quede encendido.

A parte de la Raspberry Pi necesitaremos:

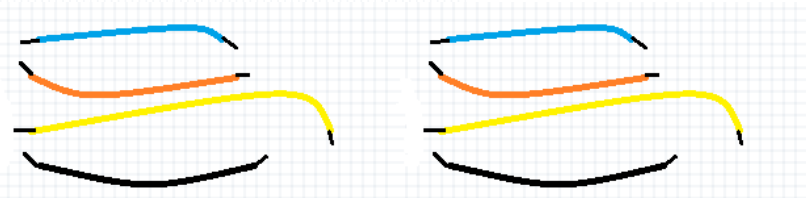
3 Leds



1 Protoboard

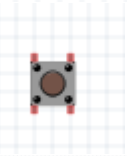


8 Cables

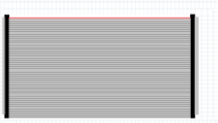


3 Resistencias

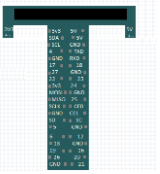
1 Botón



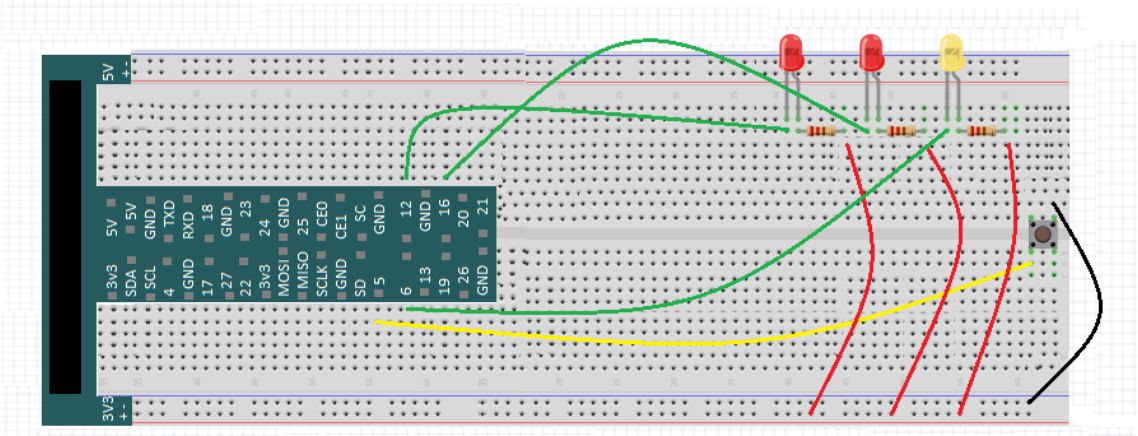
1 Bus GPIO



1 Breakout board



Montamos el circuito de la siguiente manera:



Las patas largas de los leds a una resistencia cada uno.

La pata corta del led amarillo al GPIO 5.

Las patas cortas de los leds rojos, una al GPIO 12 y otra al GPIO 16.

Una pata del botón al carril azul, el ground.

Otra pata del botón al GPIO 5.

Las resistencias al carril rojo, a la corriente de 3.3 voltios.

Ahora conectamos la Raspberry a la corriente de nuevo.

Abrimos Visual Studio para programar el código que va a hacer que al pulsar el botón se encienda y apague el led.

Creamos un proyecto igual que hemos hecho en la fase 1 y le ponemos un nombre distinto.

Al igual que en el caso anterior, dentro de la clase MainPage.xaml.cs antes del método MainPage introducimos

```
        private const int LED_Yellow_PIN = 6;
        private const int LED_Red1_PIN = 12;
        private const int LED_Red2_PIN = 16;
        private const int MaxPins = 3;
        private const int BUTTON_PIN = 5;

        private int ledEncendido = 0;

        private GpioPin ledYellowPin;
        private GpioPin ledRed1Pin;
        private GpioPin ledRed2Pin;

        private GpioPin buttonPin;

        private DispatcherTimer timer;

        private GpioPinValue ledYellowPinValue = GpioPinValue.High;
        private GpioPinValue ledRed1PinValue = GpioPinValue.High;
        private GpioPinValue ledRed2PinValue = GpioPinValue.High;

        List<GpioPin> pinValues;
```

De nuevo añadimos la directiva using que falta:

```
         using Windows.Devices.Gpio;
```

Sustituimos el método MainPage por el siguiente, encargado de iniciar el temporizador y los pines.

```
        public MainPage()
        {
            InitializeComponent();
            timer = new DispatcherTimer();
            timer.Interval = TimeSpan.FromMilliseconds(100);
            timer.Tick += Timer_Tick;
            InitGPIO();
            timer.Start();

        }
```

A continuación, creamos el método InitGPIO que se encarga de abrir los pines, establecer si son de salida (led) o de entrada (botón) y darles el valor inicial.

```
private void InitGPIO()
        {
            var gpio = GpioController.GetDefault();

            if (gpio == null)
            {
                return;
            }
            pinValues = new List<GpioPin>();


            buttonPin = gpio.OpenPin(BUTTON_PIN);
            ledYellowPin = gpio.OpenPin(LED_Yellow_PIN);
            ledRed1Pin = gpio.OpenPin(LED_Red1_PIN);
            ledRed2Pin = gpio.OpenPin(LED_Red2_PIN);

            pinValues.Add(ledYellowPin);
            pinValues.Add(ledRed1Pin);
            pinValues.Add(ledRed2Pin);

            foreach (var pin in pinValues)
            {
                pin.Write(GpioPinValue.High);
                pin.SetDriveMode(GpioPinDriveMode.Output);

            }



            if (buttonPin.IsDriveModeSupported(GpioPinDriveMode.InputPullUp))
                buttonPin.SetDriveMode(GpioPinDriveMode.InputPullUp);
            else
                buttonPin.SetDriveMode(GpioPinDriveMode.Input);


            buttonPin.DebounceTimeout = TimeSpan.FromMilliseconds(50);


            buttonPin.ValueChanged += buttonPin_ValueChanged;


        }
```

Creamos también el método `buttonPin_ValueChanged` encargado de gestionar lo que pasa al pulsar sobre el botón.

```
private async void buttonPin_ValueChanged(GpioPin sender,
GpioPinValueChangedEventArgs e)
        {
            await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal,
                () =>
                {
                    if (sender.Read() == GpioPinValue.High)
                    {
                        if (timer.IsEnabled)
                        {
                            timer.Stop();
                        }
```

```
                        else
                        {
                            timer.Start();
                        }
                    }
                });
        }
```

CoreDispatcherPriority quedará subrayado el rojo ya que debemos añadir la siguiente directiva using:

```
        using Windows.UI.Core;
```

Creamos también el método Timer_tick que se encarga de hacer que los leds se vayan encendiendo y apagando.

```
private void Timer_Tick(object sender, object e)
        {
            pinValues[ledEncendido].Write(GpioPinValue.High);
            ledEncendido++;
            if (ledEncendido== MaxPins)
            {
                ledEncendido = 0;
            }
            pinValues[ledEncendido].Write(GpioPinValue.Low);
         }
```

Ejecutamos igual que en las dos fases anteriores. Verás que los leds empiezan a encenderse y apagarse en orden. El juego consiste en pulsar el botón cuando el led amarillo esté encendido y que se quede encendido. Para aumentar la dificultad podemos disminuir el tiempo entre cada tick del temporizador para que se enciendan y apaguen más rápido. Para ello modificamos el 100 y ponemos un número más pequeño.

```
timer.Interval = TimeSpan.FromMilliseconds(100);
```